WEST Search History

Hide Items | Restore | Clear | Cancel

DATE: Friday, April 30, 2004

Hide?	Set Name	Query	Hit Count
		DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR	
☐	L34	l12 and L32	0
☐	L33	l22 and L32	0
☐	L32	l24 and L31	35
☐	L31	l27 or l28 or l29 or L30	1374
☐	L30	719/318.ccls.	381
☐	L29	719/317.ccls.	132
☐	L28	719/316.ccls.	297
☐	L27	719/315.ccls.	652
☐	L26	(L12 or L10) and L25	2
☐	L25	L23 and L24	6
☐	L24	native adj code	1053
☐	L23	L22 and L18	104
☐	L22	argument adj list	786
☐	L21	L15 and L20	0
☐	L20	L12 and L19	15
☐	L19	L10 and L18	103
☐	L18	L6 or L17 or L7	2535
☐	L17	(717/106 |717/107 |717/108 |717/109).ccls.	716
☐	L16	L13 and L15	1
☐	L15	dynamic$4 adj typ$3	3713
☐	L14	L5 and L13	0
☐	L13	L10 same L12	15
☐	L12	function adj handle$1	4514
☐	L11	L5 same L10	4
☐	L10	function adj pointer$1	1659
☐	L9	L5 and L8	84
☐	L8	L6 or L7	1997
		DB=USPT; PLUR=YES; OP=OR	
☐	L7	(717/136 |717/137 |717/138 |717/139 |717/140 |717/141 |717/142 |717/143 |717/144 |717/145 |717/146 |717/147 |717/148 |717/149 |717/150 |717/151 |717/152 |717/153 |717/154 |717/155 |717/156 |717/157 |717/158 |717/159 |717/160 |717/161).ccls.	1715
☐	L6	(717/114 |717/115 |717/116 |717/117 |717/118 |717/119).ccls.	539
☐	L5	argument adj list$1	513

/N VENTOR SEARCH DONE ON PALM 4-30-04 ST.1CL

☐	L4	L1 and L2	0
☐	L3	dynamic adj type$1	1756
☐	L2	dyanamic adj type$1	0
☐	L1	overload$3 same function$1	6324

END OF SEARCH HISTORY

First Hit

End of Result Set

☐ Generate Collection | Print

L16: Entry 1 of 1 File: PGPB Apr 3, 2003

PGPUB-DOCUMENT-NUMBER: 20030066051
PGPUB-FILING-TYPE: new
DOCUMENT-IDENTIFIER: US 20030066051 A1

TITLE: Function values in computer programming languages having dynamic types and overloading

PUBLICATION-DATE: April 3, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	COUNTRY	RULE-47
Karr, Michael	Brookline	MA	US	
Branch, Mary Ann	Waltham	MA	US	

APPL-NO: 09/ 915139 [PALM]
DATE FILED: July 25, 2001

INT-CL: [07] G06 F 15/16, G06 F 9/44, G06 F 9/00

US-CL-PUBLISHED: 717/114; 709/106
US-CL-CURRENT: 717/114; 718/106

REPRESENTATIVE-FIGURES: 2

ABSTRACT:

A method and computer program product is shown for use with a computer programming language having dynamic types and overloaded functions. A function data structure (a function handle) is constructed using a function name, which data structure contains or leads to information necessary to resolve function overloading, and also may lead to other auxiliary functions such as write, read or print. Application of the function data structure at the point of construction is functionally equivalent to application of a function name string. However, the function data structure may be applied without regard to the scope at the point of construction.

L20: Entry 1 of 15 File: PGPB Feb 13, 2003

DOCUMENT-IDENTIFIER: US 20030033589 A1
TITLE: System and method for utilization of a command structure representation

Current US Classification, US Primary Class/Subclass:
717/109

Detail Description Paragraph:
[0023] Command nodes 10-60 may be stored in a database which may include information on a particular command node 10-60 such as a keyword or name, a help string, a prompt string, the access level, "no form" capabilities, the number of child nodes, pointers to the child nodes, a number of parameters associated with the command node, pointers to the parameter descriptions, the number of handler functions, pointers to the handler functions, etc. Thus, when a developer is defining a new command there may be a multitude of pieces of information the developer may include about the new command in order for that command to be properly inserted within the command structure. The exemplary embodiment of the present invention allows the developer to quickly create CLI commands by creating the structure of command nodes. Each command node may, in turn, be edited to add parameters and handler functions. The exemplary embodiment of the present invention also allows for the automatic generation of software code for the handler functions and provides for the modification of the generated code.

Detail Description Paragraph:
[0036] Referring back to FIG. 6, when the insertion of parameter information in step 330 is complete, the process continues to step 340, where the handler functions are entered for the new command. Similar to the entering of parameters, the entering of handler functions is also accomplished via GUI 400 by adding the desired handler functions to handler function field 420. The handler functions may be added via a GUI associated with handler editor 130. GUI 400 is shown as having no handler functions in handler function field 420, but a developer may add handler functions to the new command by clicking on add button 421. Those of skill in the art will understand that a command node is not required to have a handler function. Thus, step 340 may be considered an optional step in that the developer may not elect to add any handler functions to the new command. In such a case, handler function field 420 on GUI 400 will remain empty. Delete button 413 may cause handler function field 420 to be updated if deleted parameters are associated with any handler functions.

Detail Description Paragraph:
[0037] FIG. 9 shows an exemplary GUI 500 for handler editor 130. The action of clicking on add button 421 may bring up GUI 500 which may be used by the developer to add a handler function to the new command. Those of skill in the art will understand that GUI 500 is only exemplary and that the GUI for handler editor 130 may include more or less fields than those described with reference to FIG. 9. GUI 500 may indicate the command node with which the handler function is associated (e.g., command3 node 240 in the example shown in FIG. 9). The developer may then fill in the information for the handler function via GUI 500. The developer may fill in the name of the handler function in handler name field 501 (e.g. VD_Command3Handler). A list of the available parameters will be shown in available parameter field 502. The list of available parameters will be those added by the developer in step 330 of the current command node and/or parameters in one of the parent command nodes of the current command node (i.e., parameters from any command node that is at a higher level in the same command structure branch are available to the current command node). Thus, any parameter that is added to a command node, will also be available for any child command nodes of the command node to which the parameter is added. If the parameter listed in available parameter field 502 is not associated with the current command node, but rather with a parent command node, the developer will be aware of this because it will be shown named with the command node with which it is associated. For example, an exemplary command node named command5 may have a parameter unnamed1 and a second exemplary command node named command5child, which is a child of command5, may have a parameter

unnamed 1. GUI 500 of handler editor 130 when opened for a handler function of command5child may show the list of available parameters in available parameter field 502 as:

Detail Description Paragraph:

[0053] This code describes an exemplary handler function definition array mCommand3Handlers for command3. As described above, command3 has one handler function and a pointer to the array mCommand3Handlers for the one handler function. As described above, the developer may have entered information for the handler function in, for example, GUI 500. Command structure generation engine 145 takes the information entered by the developer and generates the handler function definition file. The exemplary code above may include the following information: the type of command (e.g., can this command handle "No" forms), the bitmask of required parameters, the bitmask of optional parameters and the actual handler function associated with the definition. For example, referring to the exemplary code for the command3 handler function, the command type is "0" meaning that it cannot handle the "No" form, the bitmask of param1 is identified as a required parameter (0.times.00000001), the bitmask of param2 is identified as an optional parameter (0.times.0000002) and the actual handler function is identified as VD_Command3Handler. The actual handler function will be described in greater detail below.

L11: Entry 3 of 4 File: USPT Sep 8, 1998

DOCUMENT-IDENTIFIER: US 5805887 A
** See image for Certificate of Correction **
TITLE: Universal pointer object

Detailed Description Text (17):
The constructor function causes the computer 10 to perform the steps of blocks 42-56. Block 42 represents the creation of a variable-length argument list in the memory 14 of the computer 10. As is well known, a variable argument list includes fixed arguments which are type-checked by the compiler 22 and variable arguments which are not type-checked. According to the invention, the list includes two arguments: one fixed argument representing the type of a member function (hereinafter rule member type argument) and one variable argument representing the pointer to the member function (hereinafter rule member pointer argument). It is noted that the list may include any number of arguments, provided that the member function type is a fixed argument and the member function pointer is a variable argument.

Detailed Description Text (20):
The computer 10 then stores the member function pointer argument in a data member (hereinafter universal pointer data member) according to the member function type data member. To retrieve the member function argument from the list and store it in the universal pointer data member, the computer 10 first determines whether the member function type argument is static as indicated by decision diamond 50.

Detailed Description Text (21):
If the member function type argument is static, the member function pointer argument is retrieved from the variable argument list and stored in a static universal pointer data member as indicated by block 54. Otherwise, the member function pointer is retrieved from list and stored in a nonstatic universal pointer data member as indicated by block 52.

Detailed Description Text (23):
By passing the member function pointer argument as a variable argument of a variable-length argument list, the type and class of the pointer is not checked since the compiler only type-checks fixed arguments. Thus, a number of universal pointer objects can be created and each can hold a rule member pointer of any type and belonging to any class. Moreover, by retrieving the member function type argument prior to the member function pointer argument, the type can be used as a flag to appropriately retrieve and store the member function pointer argument.

Detailed Description Text (34):
The constructor function of the universal pointer class .sub.-- RAny.sub.-- Class is defined at lines 4-14. As discussed above and as indicated by the ellipsis (. . .) in the parameter list at line 4, the constructor function creates a variable-length argument list. According to the invention, only two arguments need to be stored by the list: a fixed argument representing the member function type and a variable argument representing the member function pointer.

Detailed Description Text (35):
As discussed above, by storing the member function pointer argument as a variable argument of a variable-length argument list, the compiler 22 does not check the type and class of the pointer. Moreover, by retrieving the member function type argument prior to the member function pointer argument, the member function type can be used as a flag to correctly retrieve and store the member function pointer.

Detailed Description Text (38):
Next, the member function pointer argument of the variable argument list is retrieved from the variable argument list and stored according to the member function type argument, as indicated at lines 9-12. To retrieve the member function pointer argument, macro va.sub.-- arg is

invoked, as indicated at lines [illegible] and 12. Macro va.sub.-- arg receives two arguments, pointer ap and the type of the value expected in the next argument in the list, and returns the value of the next argument in the list, for example, the member function pointer argument.

Detailed Description Text (39):
In the exemplary embodiment, the next argument of the list is the member function pointer argument. Thus, the second argument of va.sub.-- arg, i.e., the type of the value expected in the next argument in the list, must match the member function type argument. However, the member function type can vary. Accordingly, the present invention uses the member function type argument as a flag to correctly retrieve and store the member function pointer argument.

CLAIMS:

4. The method of claim 1, further comprising the step of creating a variable-length argument list in the memory of the computer, wherein the list includes a first argument representing the member function type and a second argument representing the member function pointer.

14. The apparatus of claim 11, further comprising means, performed by the computer, for creating a variable-length argument list in the memory of the computer, wherein the list includes a first argument representing the member function type and a second argument representing the member function pointer.

24. The program storage medium of claim 21, wherein the method further comprises the step of creating a variable-length argument list in the memory of the computer, wherein the list includes a first argument representing the member function type and a second argument representing the member function pointer.

PALM INTRANET

Day : Friday
Date: 4/30/2004
Time: 16:04:32

Continuity Information for 09/915139

Parent Data
No Parent Data

Child Data
No Child Data

Appln Info | Contents | Petition Info | Atty/Agent Info | **Continuity Data** | Foreign Data | Inventors | Address

Search Another: Application# [ ] Search **or Patent#** [ ] Search

PCT / [ ] / [ ] Search **or PG PUBS #** [ ] Search

Attorney Docket # [ ] Search

Bar Code # [ ] Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page

PALM INTRANET

Day : Friday
Date: 4/30/2004
Time: 16:04:36

Foreign Information for 09/915139

No Foreign Data

Appln Info | Contents | Petition Info | Atty/Agent Info | Continuity Data | Foreign Data | Inventors | Address

Search Another: Application# [ ] Search **or Patent#** [ ] Search

PCT / [ ] / [ ] Search **or PG PUBS #** [ ] Search

Attorney Docket # [ ] Search

Bar Code # [ ] Search

To go back use Back button on your browser toolbar.

Back to PALM | ASSIGNMENT | OASIS | Home page